### Producer-Consumer Problem

**Producer**

```plaintext
while (1) {
    produce an item in item1,
    wait (empty-space),
    wait (P-mutex),
    Buffer Linj = item1;
    in++ mod k;
    Signal (P-mutex),
    Signal (filled-space),
}
```

**Consume**

```plaintext
while (1) {
    wait (filled-space)
    wait (C-mutex),
    item2 = buffer[j] (Could)
    out++ mod k
    Signal (C-mutex),
    Signal (empty-space)
}
```

---

**Example 3**

Readers and Writers Problem

![Diagram of a Large Buffer and Reader/Writer Interaction]
Allowable Parallel Executions

> 1 Readers

Parallel operations not allowed:

R & W

w₁ & w₂  [ > 1 writer]

Writer:

Sem-type w-mutex = 1;
int num_readers = 0;

wait (w-mutex);
write to buffer

Signal (w-mutex):

Reader:

wait (r-mutex);
if (num_readers == 0) wait (w-mutex);
num_readers ++;
signal (r-mutex)

R₂ →

Read-

wait (r-mutex);
num_readers --;
if (num_readers == 0) signal (w-mutex);
signal (r-mutex)
Readers have priority over writers.

How do we fix this? **Exercise**

Problems to be aware of when using semaphores:

(a) Starvation.

(b) Deadlocks.

**Example**

```
Example P1, P2

1: wait (S1);
2: wait (S2);
3: wait (S1);
4: wait (S1);

Signal (S2);
Signal (S1);
Signal (S2);
Signal (S1);
```

Sem-type $S_1, S_2 = 1$.
Multi-threading?

9-19-13 Support multiple threads within a process.

MS-DOS: one process at any time

\[ \text{Unix:} > 1 \text{ process in the system} \]

\[ \text{Example: JVM} \]

one process
\[ > 1 \text{ thread} \]
OS's responsibilities in multi-threading.

Single Thread Case:

TCB - has PC, Register values, etc.
Thread has

1. State (Running, Ready ...)
2. Thread Context
3. Stack
4. Per-thread storage

Access to memory, resources (files ...) shared with other threads. One thread makes a change to a var, other threads will see it.

Why multi-threading?

1. CPU utilization
   Thread switching is faster than process switching

2. >1 CPU? Can speedup

3. Modular Code
   - Server code: new request?
     Give to new thread.
   - Background - foreground tasks.
   - Asynchronous processing
Thread Operations

- Spawn - Create new thread. 1st thread of a process created automatically.

- Block

- Unblock

- Terminate

Thread Synchronization?

Like processes use semaphores.

Thread Implementation

A. User Level Threads

- OS does not support.

Thread Library

user

OS (Kernel)
Thread Library has code for all thread operations:
- Creation, switching, scheduling, messaging...

Good:
1. Thread operations don't need kernel level switch
2. Can run on any OS.
3. Thread scheduling can be customized.

Disadv:
1. >1 CPU?
2. One thread blocks?

B. Kernel Level Threads

C. Combined